1] Write a program to search for an element in an array using binary and linear search:

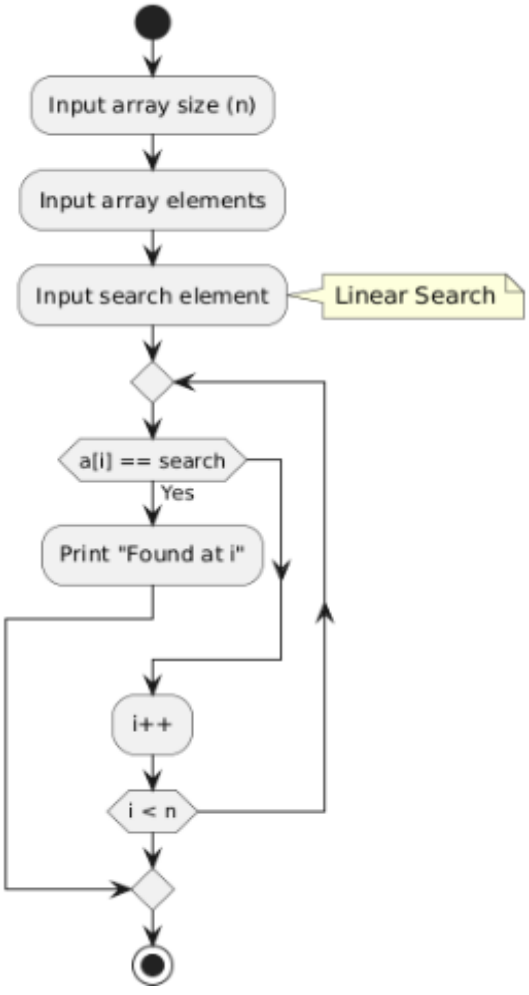
```
#include <stdio.h>
#include<conio.h>
void main()
{
 int a[10],n,search,i,j,low,high,mid,temp;
 printf("Enter size of array\n");
 scanf("%d",&n);
 printf("Enter array elements\n");
 for(i=0;i<n;i++)
 scanf("%d",&a[i]);
 printf("Enter element to be searched\n");
 scanf("%d",&search);
 //linear search
 printf("****USING LINEAR SEARCH****\n\n");
 for (i = 0; i< n; i++)
 {
 if(a[i] == search)
 {
 printf("Element %d found at location %d\n",search,i) ;
 }
 }
 // binary search
printf("\n****USING BINARY SEARCH****\n\n");
 // sorting given array
 for(i=0;i<n;i++)
 {
 for(j=0;j<n-i-1;j++)
 {
 if(a[j]>a[j+1])
 {
 temp=a[j];
 a[j]=a[j+1];
 a[j+1]=temp;
 }
 }
 }
 low=0,high=n;
 while(low<=high)
 {
 mid=(low+high)/2;
 if(a[mid]==search)
 {
 printf("Element %d found at %d location\n",search,mid);
 }
 if(search>a[mid])
```

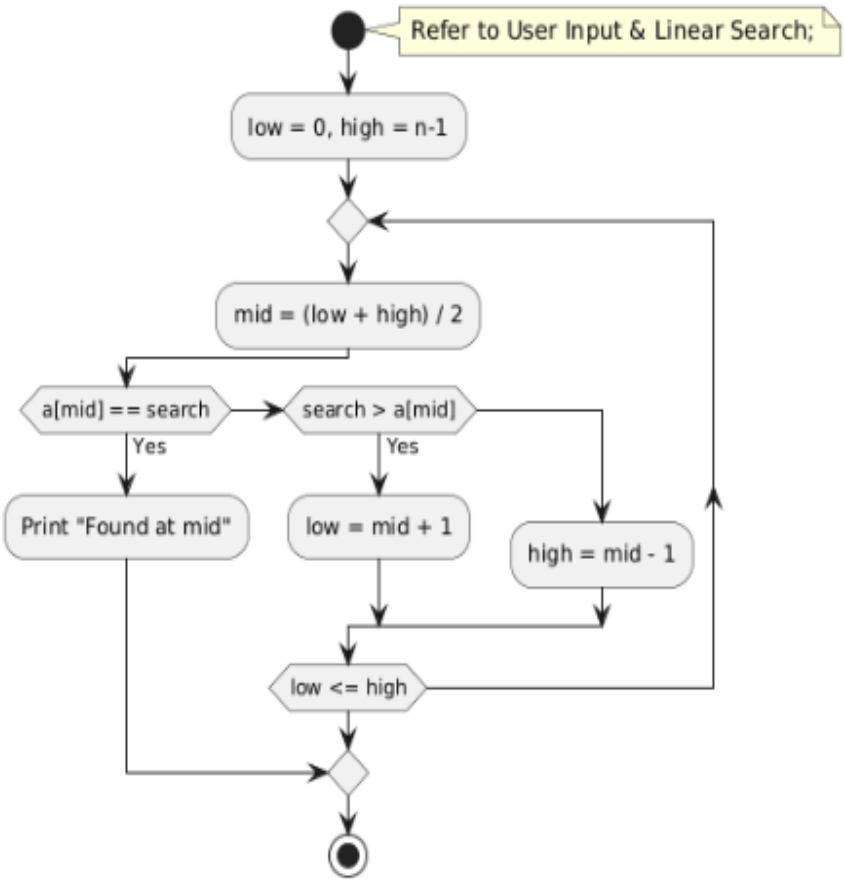
```
 low=mid+1;
 else
 high=mid-1;
 }
getch();
}
```

Output:

```
Enter size of array: 4
Enter array elements:
6
4
8
2
Enter element to be searched: 4

**** USING LINEAR SEARCH ****
Element 4 found at index 1

**** USING BINARY SEARCH ****
Element 4 found at index 1
```

Flowchart:

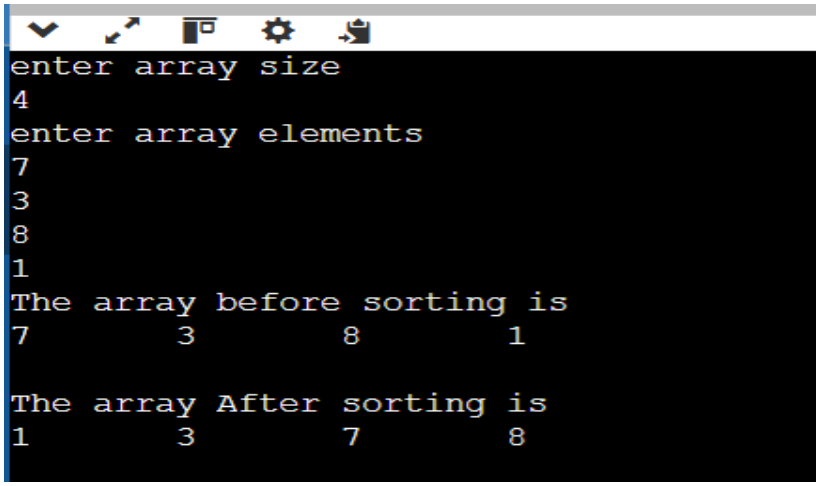



Refer to User Input & Linear Search;
low = 0, high = n-1
mid = (low + high) / 2
a[mid] == search
Yes
Print "Found at mid"
search > a[mid]
Yes
low = mid + 1
high = mid - 1
low <= high

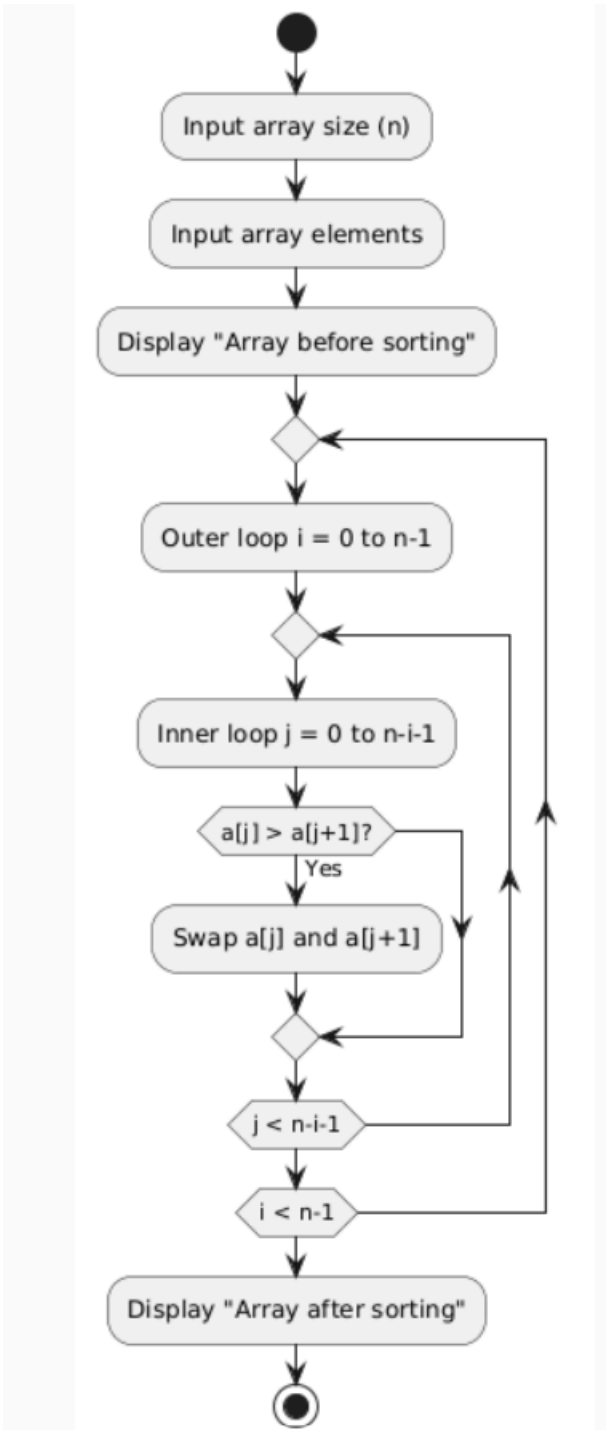
2] Write a program to sort list of n numbers using Bubble sort algorithm:

```
#include <stdio.h>
#include<conio.h>
int main()
{
 int a[20],i,j,n,temp;
 printf("enter array size\n");
 scanf("%d",&n);
 printf("enter array elements\n");
 for(i=0;i<n;i++)
 scanf("%d",&a[i]);
 printf("The array before sorting is\n");
 for(i=0;i<n;i++)
 printf("%d\t",a[i]);
 for(i=0;i<n;i++)
 {
 for(j=0;j<n-i-1;j++)
 {
 if(a[j]>a[j+1])
 {
 temp=a[j];
 a[j]=a[j+1];
 a[j+1]=temp;
 }
 }
 }
 printf("\n\nThe array After sorting is\n");
 for(i=0;i<n;i++)
 printf("%d\t",a[i]);
 getch();
}
```

Output:


enter array size
4
enter array elements
7
3
8
1
The array before sorting is
7 3 8 1
The array After sorting is
1 3 7 8

Flowchart:

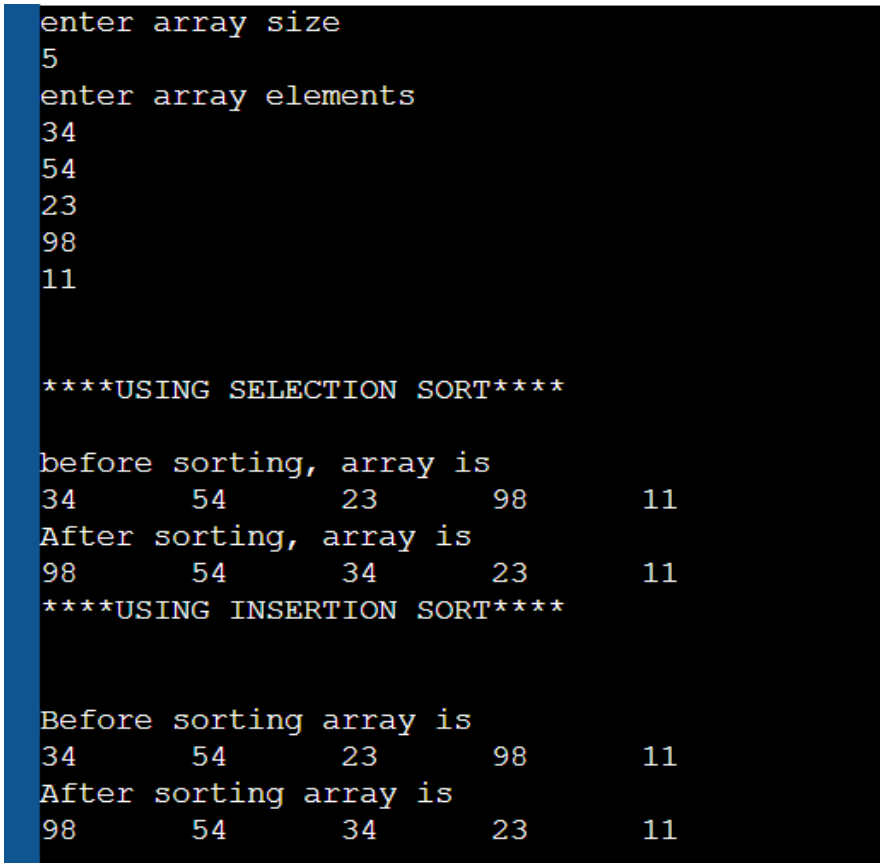

Input array size (n)
Input array elements
Display "Array before sorting"
Outer loop i = 0 to n-1
Inner loop j = 0 to n-i-1
a[j] > a[j+1]?
Yes
Swap a[j] and a[j+1]
j < n-i-1
i < n-1
Display "Array after sorting"

3] Perform the Insertion and Selection Sort on the input {75,8,1,16,48,3,7,0} and display the output in descending order:

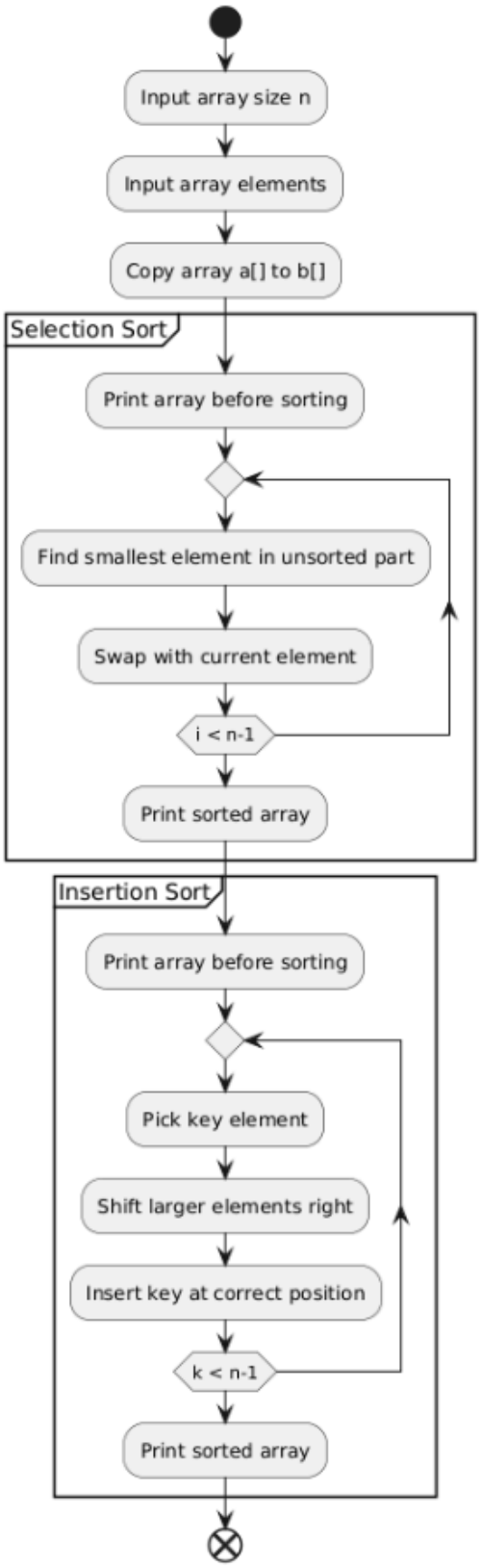
```
#include <stdio.h>
#include<conio.h>
void main()
{
 int a[10],b[10],n,i,j,k,small,loc,key;
 printf("enter array size \n");
 scanf("%d",& n);
 printf("enter array elements\n");
 for(i=0;i<n;i++)
 scanf("%d", & a[i]);
 //copying the given array
 for(i=0;i<n;i++)
 {
 b[i]=a[i];
 }
 //Selection sort
 printf("\n\n****USING SELECTION SORT****\n\n");
 printf("before sorting, array is\n");
 for(i=0;i<n;i++)
 printf("%d\t",a[i]);
 for(i=0;i<n;i++)
 {
 small=a[i];
 loc=i;
 for(j=i+1;j<n;j++)
 {
 if(a[j]<small)
 {
 small=a[j];
 loc=j;
 }
 }
 a[loc]=a[i];
 a[i]=small;
 }
 printf("\nAfter sorting, array is\n");
 for(i=n-1;i>=0;i--)
 printf("%d\t",a[i]);
 //Insertion Sort
 printf("\n****USING INSERTION SORT****\n\n");
 printf("\nBefore sorting array is \n");
 for(i=0;i<n;i++)
 printf("%d\t",b[i]);
 for(k=1;k<n;k++)
```

```
{
key=b[k];
for(i=k-1;i>=0 && key<b[i];i--)
{
b[i+1]=b[i];
b[i]=key;
}
}
printf("\nAfter sorting array is \n");
for(i=n-1;i>=0;i--)
printf("%d\t",b[i]);
getch();
}
```

Output:



Flowchart:


Input array size n
Input array elements
Copy array a[] to b[]
Selection Sort
Print array before sorting
Find smallest element in unsorted part
Swap with current element
i < n-1
Print sorted array
Insertion Sort
Print array before sorting
Pick key element
Shift larger elements right
Insert key at correct position
k < n-1
Print sorted array

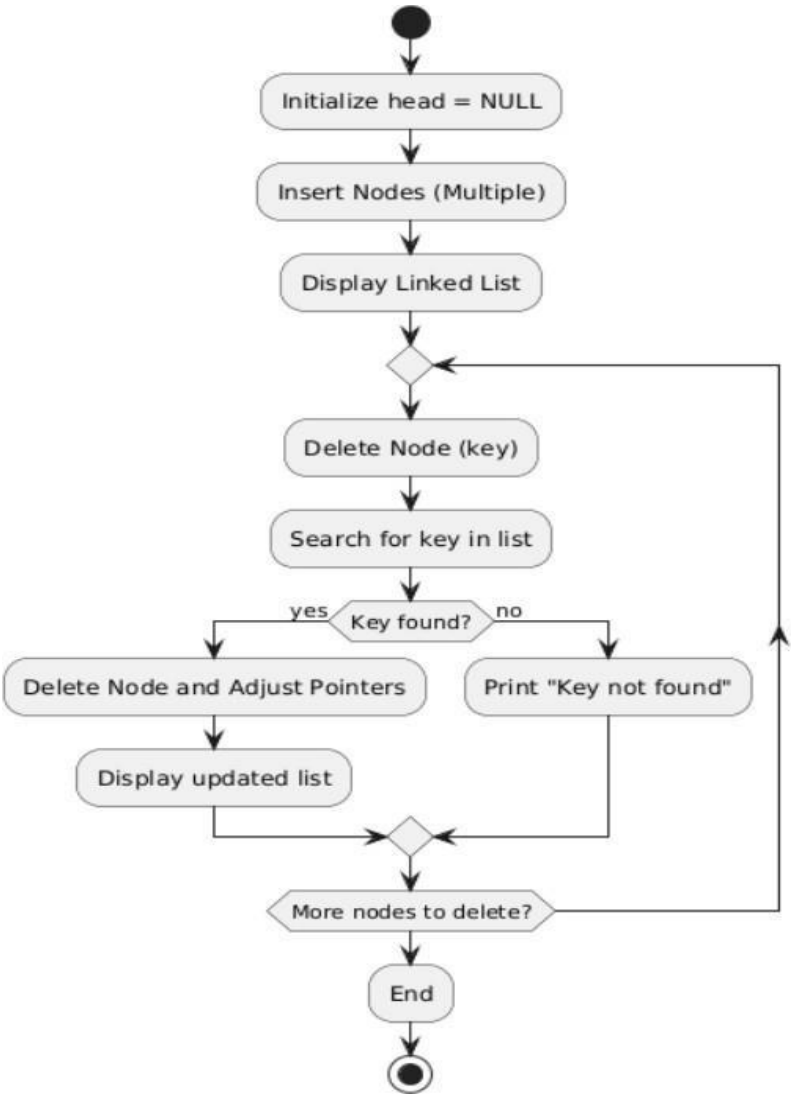
Program 4. Write the program to insert the elements{61,16,8,27} into singly Linked List and delete 8,61,27 from the List Display your list after each insertion and deletion.

```
#include  <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node* next;
};

void insert(struct Node** head, int data)
{
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = *head;
   *head = newNode;
   printf("Inserted: %d\n", data);
}

void deleteNode(struct Node** head, int key)

 {

struct Node* temp = *head, *prev = NULL;
   if (temp != NULL && temp->data == key)
    {
      *head = temp->next;
      free(temp);
      printf("Deleted: %d\n", key);
      return;
   }
   while (temp != NULL && temp->data != key)
    {
    prev = temp;
      temp = temp->next;
```

```
    }
    if (temp == NULL) return;
    prev->next = temp->next;

    free(temp);
    printf("Deleted: %d\n", key);
}

void display(struct Node* head)
{ while (head)
{
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

int main()
 {
    struct Node* head = NULL;
    insert(&head, 61);
    insert(&head, 16);
    insert(&head, 8);
    insert(&head, 27);
    display(head);
    deleteNode(&head, 8);
    display(head);
    deleteNode(&head, 61);
    display(head);
    deleteNode(&head, 27);
    display(head);
    return 0;
```

output:

```
Inserted: 61
Inserted: 16
Inserted: 8
Inserted: 27
27 -> 8 -> 16 -> 61 -> NULL
Deleted: 8
27 -> 16 -> 61 -> NULL
Deleted: 61
27 -> 16 -> NULL
Deleted: 27
16 -> NULL
```



5] Write a program to insert elements into a linear queue and delete elements.

```
/* Write a program to insert the elements {61,16,8,27} into linear queue and delete three elements f
rom the
list. Display your list after each insertion and deletion.  */

#define MAX 4
#include<stdio.h>
int front = 0, rear = -1;
int queue[MAX];
main()
{
void qinsert(int);
void qdisplay();
void qdelete();
int item, choice;
clrscr();
while(1)
{
printf("\n Queue simulator:");
printf("\n 1. ADD");
printf("\n 2. DISPLAY");
printf("\n 3. DELETE");
printf("\n 4. EXIT");
printf("\n ENTER YOUR CHOICE");
scanf("%d", &choice);
switch(choice)
{
case 1:qinsert(item);
       break;
case 2:printf("\n ELEMENTS IN THE QUEUE");
       qdisplay();
       break;
case 3:qdelete();
       break;
case 4:exit();
       default : printf("\n ERROR IN CHOICE");
}
}
}

void qinsert(int item)
{
if (isfull())
{
printf("\n QUEUE OVERFLOW");
return;
}
else
{
printf("\n Enter the elements to be added");
scanf("%d",&item);
rear++;
queue[rear] = item;
if(rear == MAX -1)
printf("\n Queue is full");
}
return;
}

void qdisplay()
{
int p;
if (isempty())
```

```
{
printf("QUEUE IS EMPTY \n");
return;
}
p = front;
while (p <= rear)
{
printf("\n%d",queue[p]);
p++;
}
return;
}

void qdelete()
{

if (isempty())
{
printf("\n QUEUE UNDERFLOW");
return;
}
else
{
int item;
item = queue[front];
front++;
printf("\n The deleted item is %d",item);
if(front==rear+1)
printf("\nQueue is empty");
}
return;
}

int isfull()
{
if(rear == MAX -1)
return 1;
else
return 0;
}

int isempty()
{
if (front==rear + 1)
return 1;
else
return 0;
}
```

6] Write a program to simulate the working of Circular Queue using an Array:

```
#include <stdio.h>
#define SIZE 4

int front = -1, rear = -1;
int queue[SIZE];

void display_CQ() {
    int i;
    if (front == -1) {
        printf("\nCircular Queue is empty.");
        return;
    }

    printf("\nCircular Queue: ");
    if (front <= rear) {
        for (i = front; i <= rear; i++)
            printf("%d -> ", queue[i]);
    } else {
        for (i = front; i < SIZE; i++)
            printf("%d -> ", queue[i]);
        for (i = 0; i <= rear; i++)
            printf("%d -> ", queue[i]);
    }
    printf("[%d]\n", queue[front]);  // Indicating the front of the queue
}

void insert_CQ(int item) {
    if ((rear + 1) % SIZE == front) {  // Full condition
        printf("\nQueue is full.");
        return;
    }

    if (front == -1) {  // First element insertion
        front = 0;
        rear = 0;
    } else {
        rear = (rear + 1) % SIZE;  // Circular increment
    }
    queue[rear] = item;
    printf("\nInserted: %d", item);
    display_CQ();
}

void delete_CQ() {
```

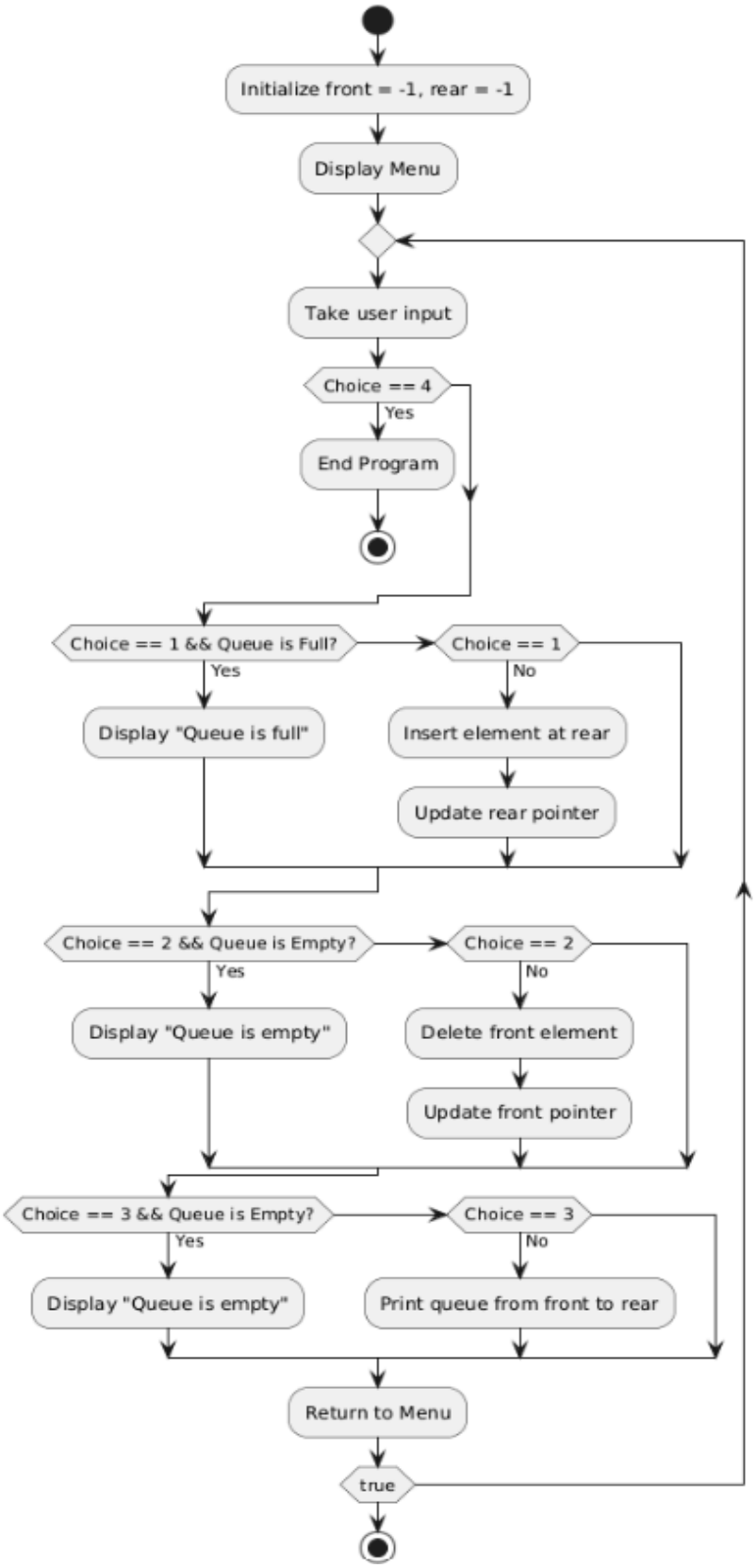
```
    if (front == -1) {  // Empty queue
        printf("\nQueue is empty.");
        return;
    }

    printf("\nDeleted: %d", queue[front]);

    if (front == rear) {  // Last element being deleted
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % SIZE;  // Circular increment
    }
    display_CQ();
}

int main() {
    int choice, value;

    while (1) {
        printf("\n\nCircular Queue Menu:");
        printf("\n1. Insert");
        printf("\n2. Delete");
        printf("\n3. Display");
        printf("\n4. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert_CQ(value);
                break;
            case 2:
                delete_CQ();
                break;
            case 3:
                display_CQ();
                break;
            case 4:
                printf("\nExiting the program...\n");
                return 0;
            default:
                printf("\nInvalid choice! Please try again.");
        }
    }
}
```

Output:

```
Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 56

Inserted: 56
Circular Queue: 56 -> [56]


Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 33

Inserted: 33
Circular Queue: 56 -> 33 -> [56]


Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 76

Inserted: 76
Circular Queue: 56 -> 33 -> 76 -> [56]
```

```
Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2

Deleted: 56
Circular Queue: 33 -> 76 -> [33]


Circular Queue Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4

Exiting the program...
```

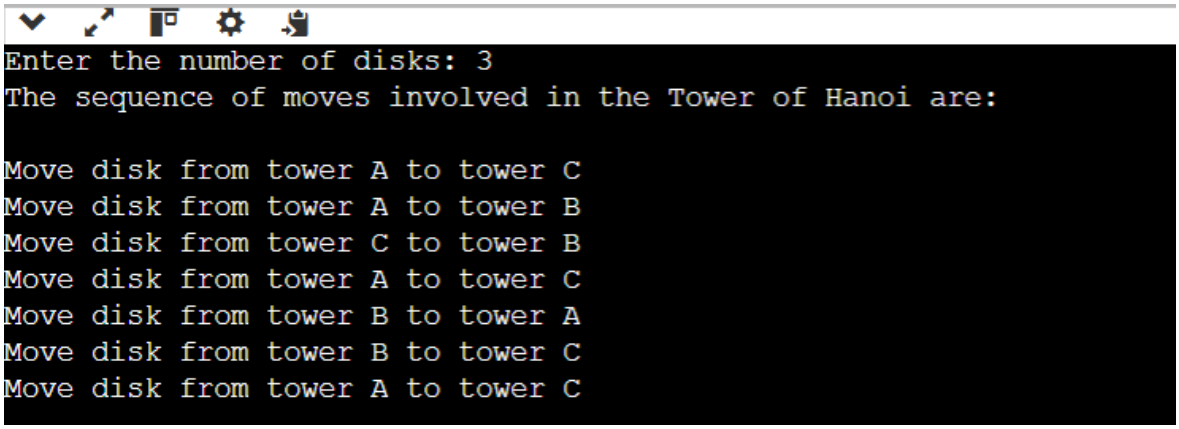
Flowchart:


Initialize front = -1, rear = -1
Display Menu
Take user input
Choice == 4
Yes
End Program
Choice == 1 && Queue is Full?
Yes
Display "Queue is full"
Choice == 1
No
Insert element at rear
Update rear pointer
Choice == 2 && Queue is Empty?
Yes
Display "Queue is empty"
Choice == 2
No
Delete front element
Update front pointer
Choice == 3 && Queue is Empty?
Yes
Display "Queue is empty"
Choice == 3
No
Print queue from front to rear
Return to Menu
true

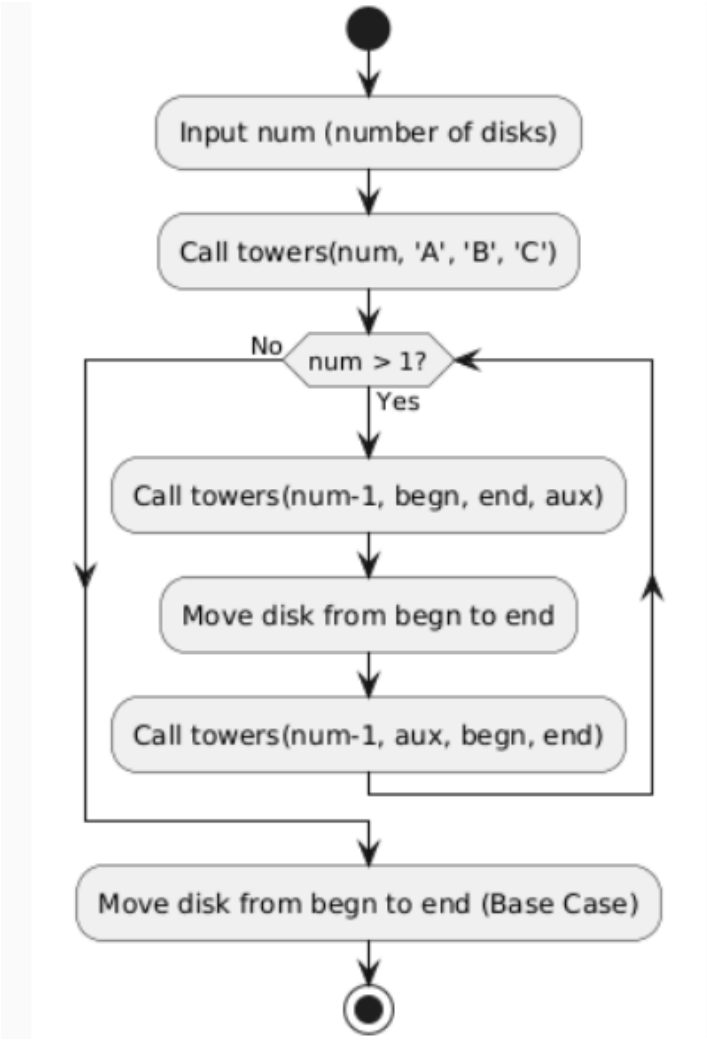
8] Write a program for Tower of Hanoi problem using recursion:

```
#include <stdio.h>

void towers(int num, char begn, char aux, char end);

int main()
 {
    int num;
    printf("Enter the number of disks: ");
    scanf("%d", &num);
    printf("The sequence of moves involved in the Tower of Hanoi are:\n");
    towers(num, 'A', 'B', 'C');  // Using uppercase letters for clarity
    return 0;
}
void towers(int num, char begn, char aux, char end) {
    if (num == 1) {
        printf("\nMove disk from tower %c to tower %c", begn, end);
        return;
    }
    towers(num - 1, begn, end, aux);  // Move n-1 disks to auxiliary tower
    printf("\nMove disk from tower %c to tower %c", begn, end);  // Move nth disk
    towers(num - 1, aux, begn, end);  // Move n-1 disks from auxiliary to destination
}
```

Output:


Enter the number of disks: 3
The sequence of moves involved in the Tower of Hanoi are:
Move disk from tower A to tower C
Move disk from tower A to tower B
Move disk from tower C to tower B
Move disk from tower A to tower C
Move disk from tower B to tower A
Move disk from tower B to tower C
Move disk from tower A to tower C

Flowchart:


Input num (number of disks)
Call towers(num, 'A', 'B', 'C')
No
num > 1?
Yes
Call towers(num-1, begn, end, aux)
Move disk from begn to end
Call towers(num-1, aux, begn, end)
Move disk from begn to end (Base Case)

9.Write a recursive program to find GCD of three Numbers.

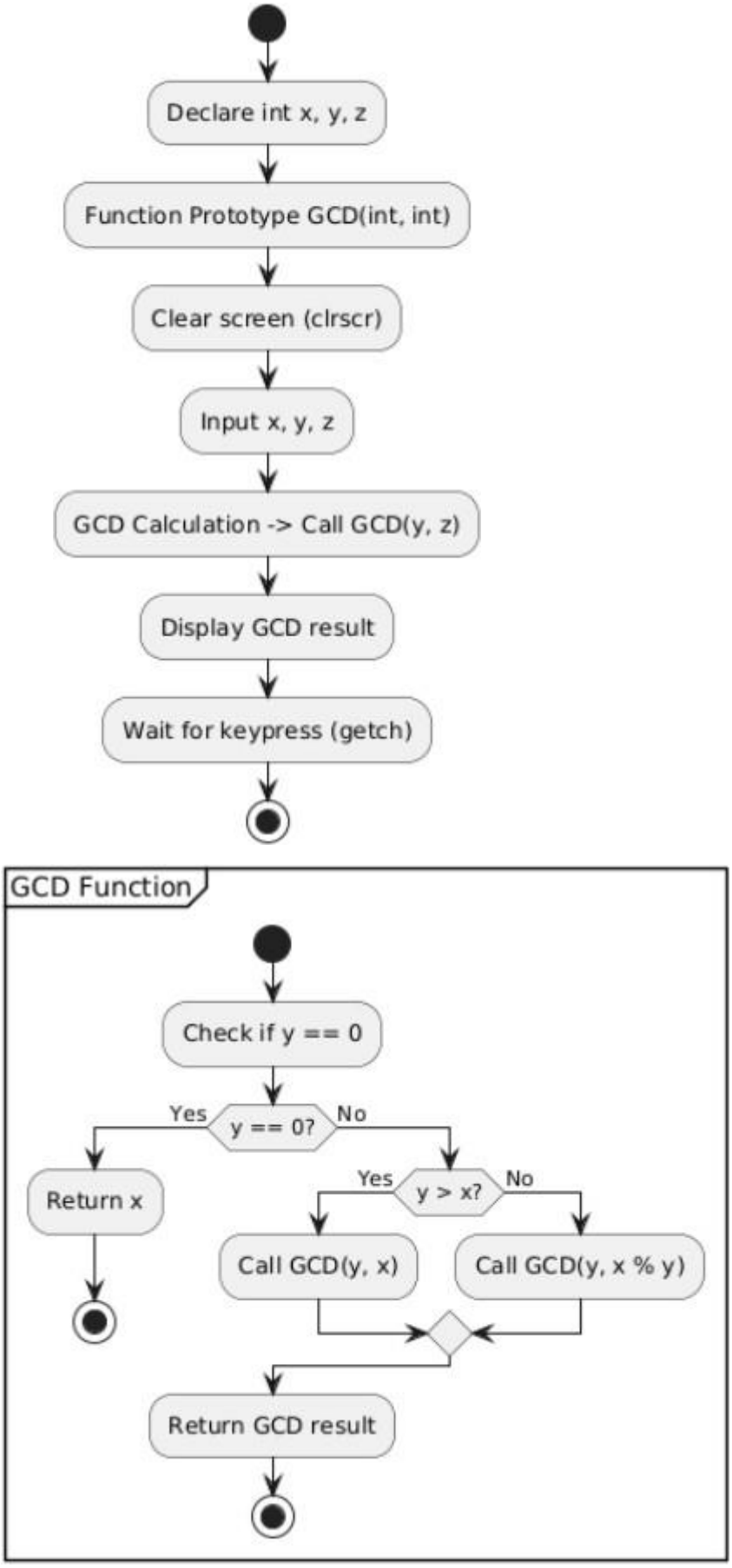
```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int x,y,z;
int GCD(int,int);
clrscr();
printf("Enter three numbers:\n");
scanf("%d%d%d",&x,&y,&z);
printf("GCD(%d,%d,%d)=%d\n",x,y,z,(x,GCD(y,z)));
getch();
}
int GCD(int x,int y)
{
if(y==0)
return(x);
else if(y>x)
return(GCD(y,x));
else
return(GCD(y,(x%y)));
}
```

```
Enter three numbers:
4 6 8
```

Output:

```
mathematica

GCD(4, 6, 8) = 2
```

Flowchart:


Declare int x, y, z
Function Prototype GCD(int, int)
Clear screen (clrscr)
Input x, y, z
GCD Calculation -> Call GCD(y, z)
Display GCD result
Wait for keypress (getch)
GCD Function
Check if y == 0
Yes
y == 0?
No
Return x
Yes
y > x?
No
Call GCD(y, x)
Call GCD(y, x % y)
Return GCD result

Program 10. Write a Program to demonstrate working of stack using Linked List

```
#include  <stdio.h>
#include <stdlib.h>
#include <string.h>

// Q10: Stack using Linked List
struct Node {
   int data;
   struct Node* next;
};

struct Node* top = NULL;

void push(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
newNode->next = top;
 top = newNode;
printf("Pushed %d onto stack\n", data);
}

void pop() {
   if (top == NULL) {
      printf("Stack Underflow\n");
      return;
   }
   struct Node* temp = top;
   printf("Popped %d from stack\n", temp->data);
   top = top->next;
   free(temp);
}

void displayStack() {
   struct Node* temp = top;
   printf("Stack: ");
   while (temp) {
      printf("%d ", temp->data);
      temp = temp->next;
   }
```

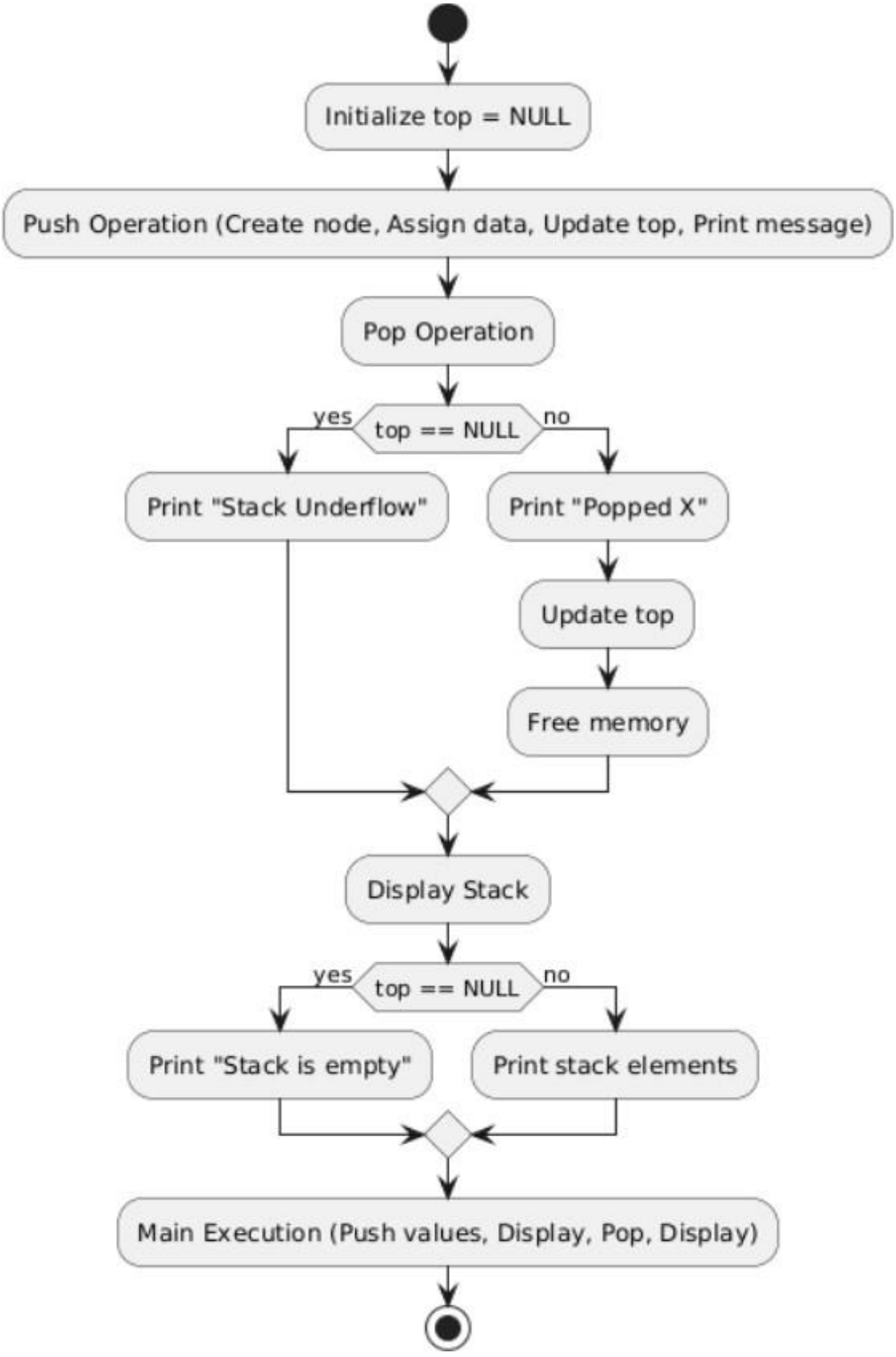
```
    printf("\n");
}

int main() {
    printf("Stack using Linked List:\n");
    push(10);
    push(20);
    push(30);
    displayStack();
    pop();
    displayStack();
    return 0;
}
```

Input:

```c
Stack using Linked List:
push(10);
push(20);
push(30);
displayStack();
pop();
displayStack();
```

Output:

```cpp
Stack using Linked List:
Pushed 10 onto stack
Pushed 20 onto stack
Pushed 30 onto stack
Stack: 30 20 10
Popped 30 from stack
Stack: 20 10
```


Initialize top = NULL
Push Operation (Create node, Assign data, Update top, Print message)
Pop Operation
yes
top == NULL
no
Print "Stack Underflow"
Print "Popped X"
Update top
Free memory
Display Stack
yes
top == NULL
no
Print "Stack is empty"
Print stack elements
Main Execution (Push values, Display, Pop, Display)

11. Write a program to convert an infix expression x^y/(5*2)+2 to its postfix expression.

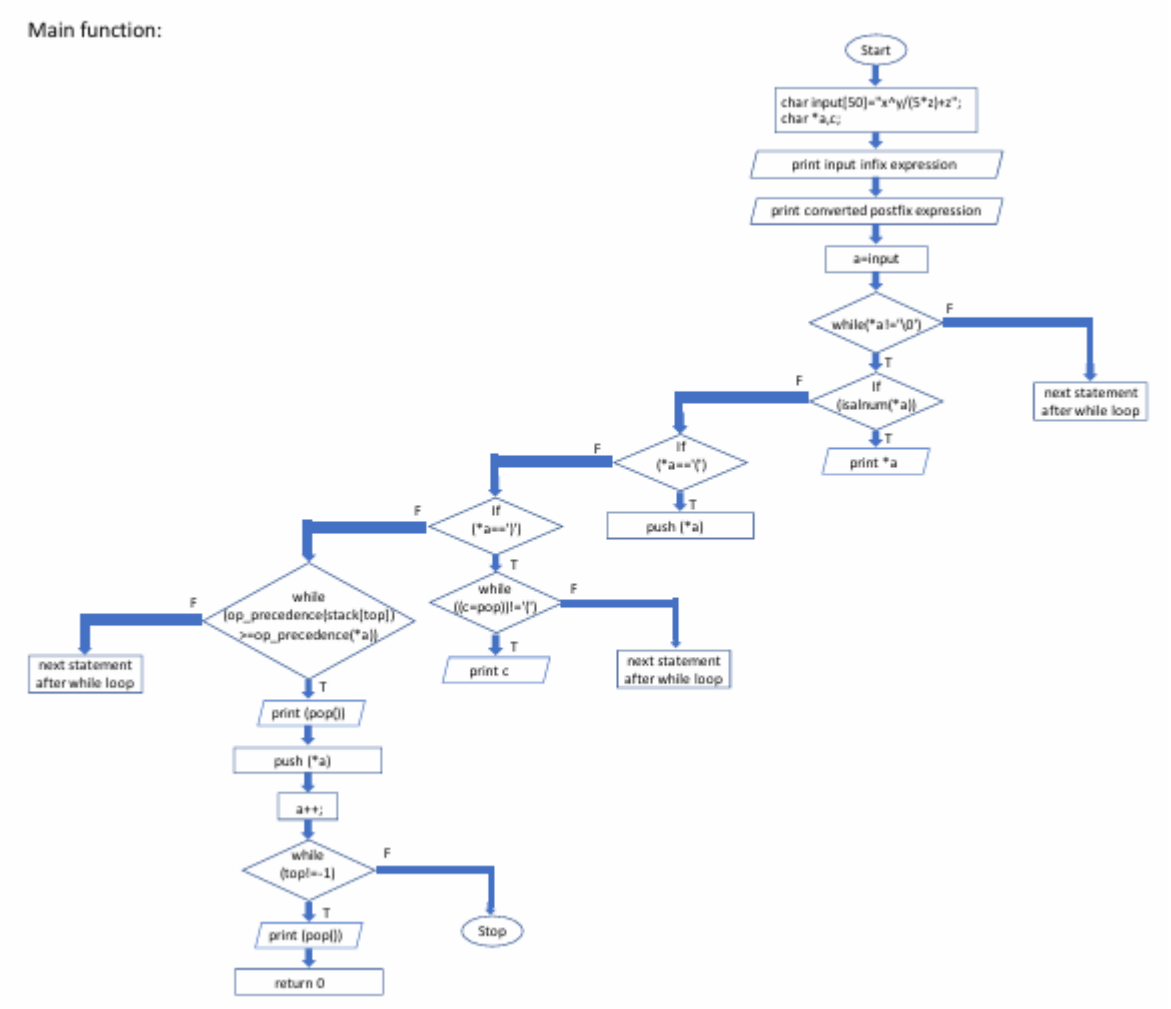
```
#include<stdio.h>
#include<ctype.h>
char stack[50];
int top=-1;
// assigning the operator precedence
int op_precedence(char c)
{
if(c == '(')
return 0;
if(c == '+' || c == '-')
return 1;
if(c == '*' || c == '/')
return 2;
if(c == '^')
return 3;
return 0;
}

void push(char c)
{
stack[++top] = c;
}

char pop()
{
if(top == -1)
return -1;
else
return stack[top--];
}
```

```
int main()
{
char input[50] = "x^y/(5*z)+2";
char *a,c;
clrscr();
printf("Input infix expression: %s\n",input);
printf("Converted postfix expression: ");
a = input;
while(*a != '\0')
{
if(isalnum(*a))
printf("%c",*a);
else
if(*a == '(')
push(*a);
else if(*a == ')')
{
while((c=pop()) != '(')
printf("%c",c);
}
else
{
while(op_precedence(stack[top]) >=op_precedence(*a))
printf("%c",pop());
push(*a);
}
a++;
}
while(top != -1)
{
```

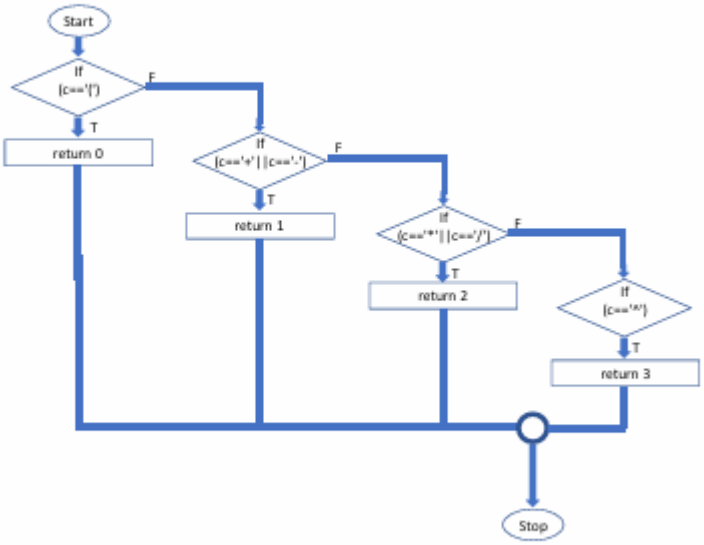
```
printf("%c",pop());

}

return 0;

}
```

Output:

Flowchart:

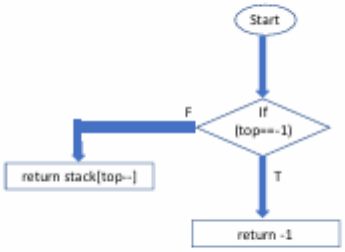


Int op_precedence(char c) function:



Push function:

Pop function:



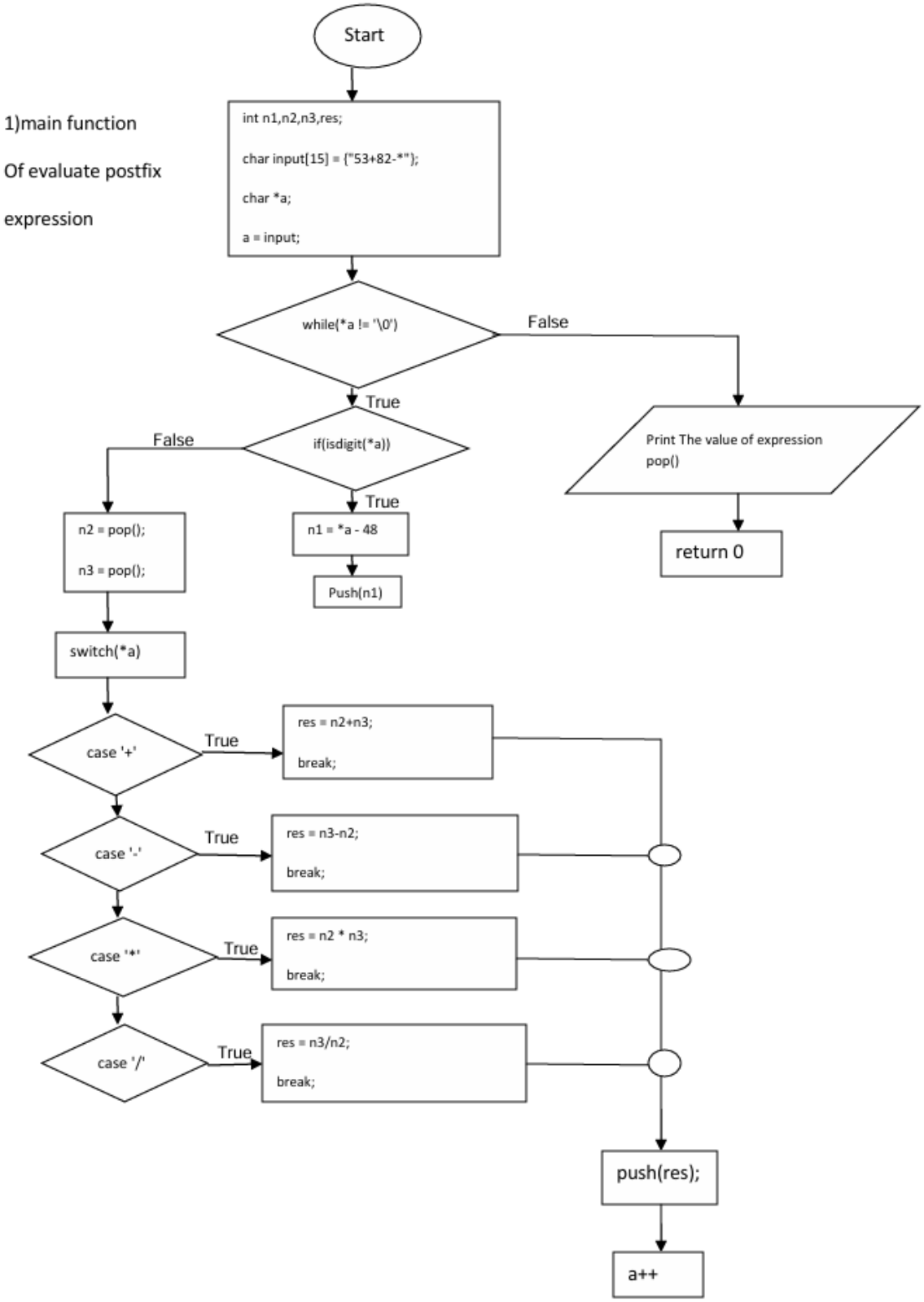
12. Write a program to evaluate a postfix expression 53+82-*.

```
#include<stdio.h>
#include<ctype.h>
int stack[10];
int top = -1;

void push(int value)
{
stack[++top] = value;
}

int pop()
{
int value = stack[top];
top--;
return value;
}

int main()
{
int n1,n2,n3,res;
char input[15] = {"53+82-*"};
char *a;
a = input;
while(*a != '\0')
{
if(isdigit(*a))
{
//to convert a character digit to integer
n1 = *a - 48;
push(n1);
```

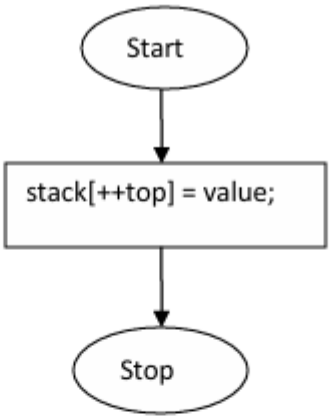
```
}
else
{
n2 = pop();
n3 = pop();
switch(*a)
{
case '+':res = n2+n3;
break;
case '-':res = n3-n2;
break;
case '*':res = n2 * n3;
break;
case '/':res = n3/n2;
break;
}
push(res);
}
a++;
}
printf("\nThe value of expression 53+82-* after evaluation is: %d\n",pop());
return 0;
}
```

Output:

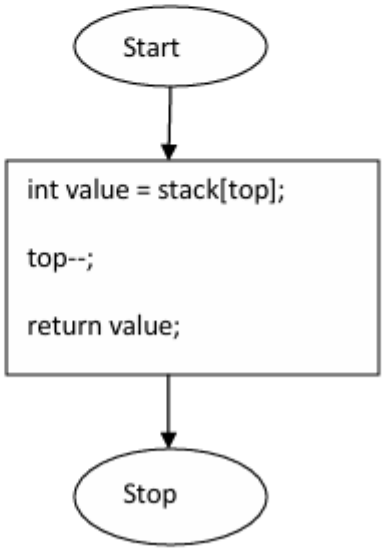
Flowchart:


Start
1)main function
Of evaluate postfix
expression
int n1,n2,n3,res;
char input[15] = {"53+82-*"};
char *a;
a = input;
while(*a != '\0')
False
True
Print The value of expression
pop()
return 0
if(isdigit(*a))
False
True
n2 = pop();
n3 = pop();
n1 = *a - 48
Push(n1)
switch(*a)
case '+'
True
res = n2+n3;
break;
case '-'
True
res = n3-n2;
break;
case '*'
True
res = n2 * n3;
break;
case '/'
True
res = n3/n2;
break;
push(res);
a++

2)push() function :



3)pop() function:



14. Write a program to create binary search tree with the elements (2,5,1,3,9,0,6) and perform inorder, preorder and post order traversal.

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
int data;
struct Node* left;
struct Node* right;
};
void insertNode(struct Node** subtree,int key)
{
	struct Node* temp = NULL;
	if(!(*subtree))
	{
		temp = (struct Node*)malloc(sizeof(struct Node));
		temp->data = key;
		temp->left = NULL;
		temp->right = NULL;
		*subtree = temp;
		return;
	}
	if(key<(*subtree)->data)
	{
		insertNode(&(*subtree)->left,key);
	}
	else if(key>(*subtree)->data)
	{
		insertNode(&(*subtree)->right,key);
	}
}
struct Node* searchNode(struct Node** subtree,int key)
```

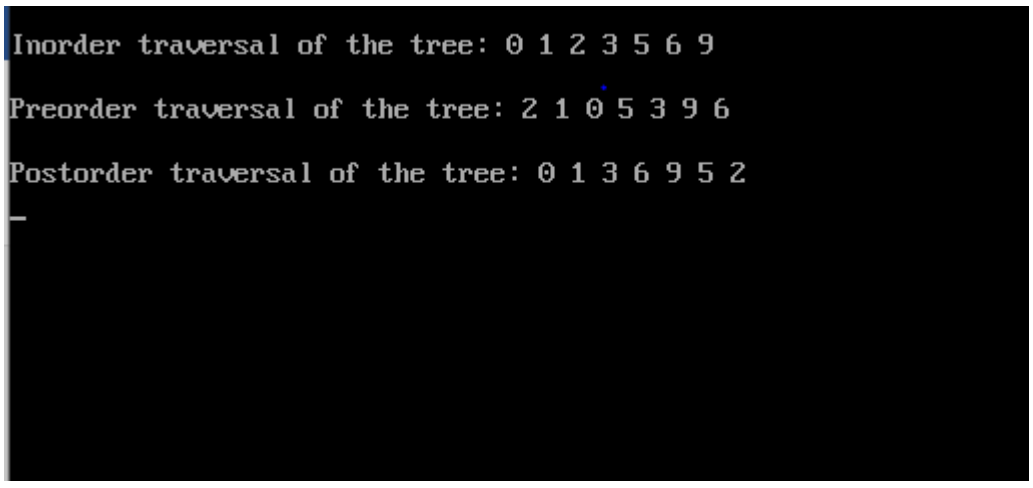
```
{
	if(!(*subtree))
	{
		return NULL;
	}
	if(key<(*subtree)->data)
	{
		searchNode(&(*subtree)->left,key);
	}
	else if(key>(*subtree)->data)
	{
		searchNode(&(*subtree)->right,key);
	}
	else if(key == (*subtree)->data)
	{
		return *subtree;
	}
}
struct Node* minValue(struct Node* node)
{
	struct Node* temp = node;
	while(temp->left != NULL)
	{
		temp = temp->left;
	}
	return temp;
}
struct Node* deleteNode(struct Node* root,int key)
{
	if(root == NULL)
	{
```

```
            return root;
        }
        if(key < root->data)
        {
            root->left = deleteNode(root->left,key);
        }
        else if(key > root->data)
        {
            root->right = deleteNode(root->right,key);
        }
        else
        {
            if(root->left == NULL)
            {
                struct Node* temp = root->right;
                free(root);
                return temp;
            }
            else if(root->right == NULL)
            {
                struct Node* temp = root->left;
                free(root);
                return temp;
            }
            {
                struct Node* temp = minValue(root->right);
                root->data = temp->data;
                root->right = deleteNode(root->right,temp->data);
            }
        }
return root;
```

```
}
void inorder(struct Node* tree){ //left,root,right
if(tree){
inorder(tree->left);
printf("%d ",tree->data);
inorder(tree->right);
}
}
void preorder(struct Node* tree){ //root,left,right
if(tree){
printf("%d ",tree->data);
preorder(tree->left);
preorder(tree->right);
}
}
void postorder(struct Node* tree){ //left,right,root
if(tree){
postorder(tree->left);
postorder(tree->right);
printf("%d ",tree->data);
}

}
int main(){
struct Node* root = NULL;
clrscr();
insertNode(&root,2);
insertNode(&root,5);
insertNode(&root,1);
insertNode(&root,3);
insertNode(&root,9);
```

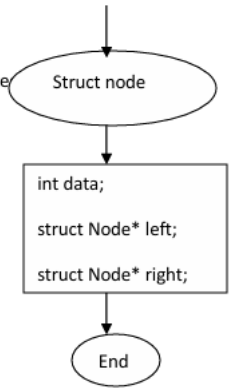
```
insertNode(&root,0);
insertNode(&root,6);
printf("\n");
printf("Inorder traversal of the tree: ");
inorder(root);
printf("\n\n");
printf("Preorder traversal of the tree: ");
preorder(root);
printf("\n\n");
printf("Postorder traversal of the tree: ");
postorder(root);
printf("\n");
return 0;
}
```

Output:

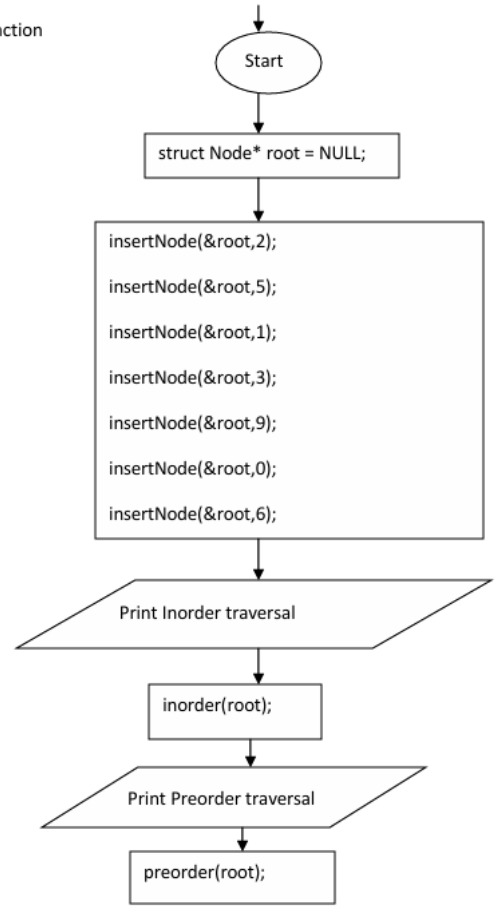


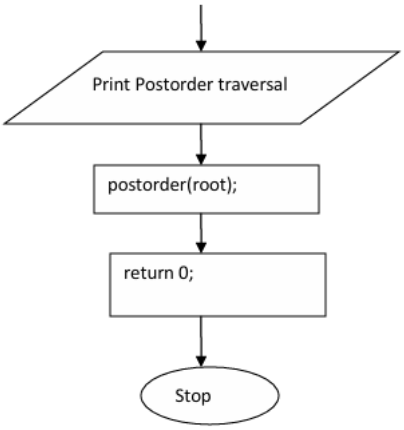
Flowchart:

1) to create binary search tree

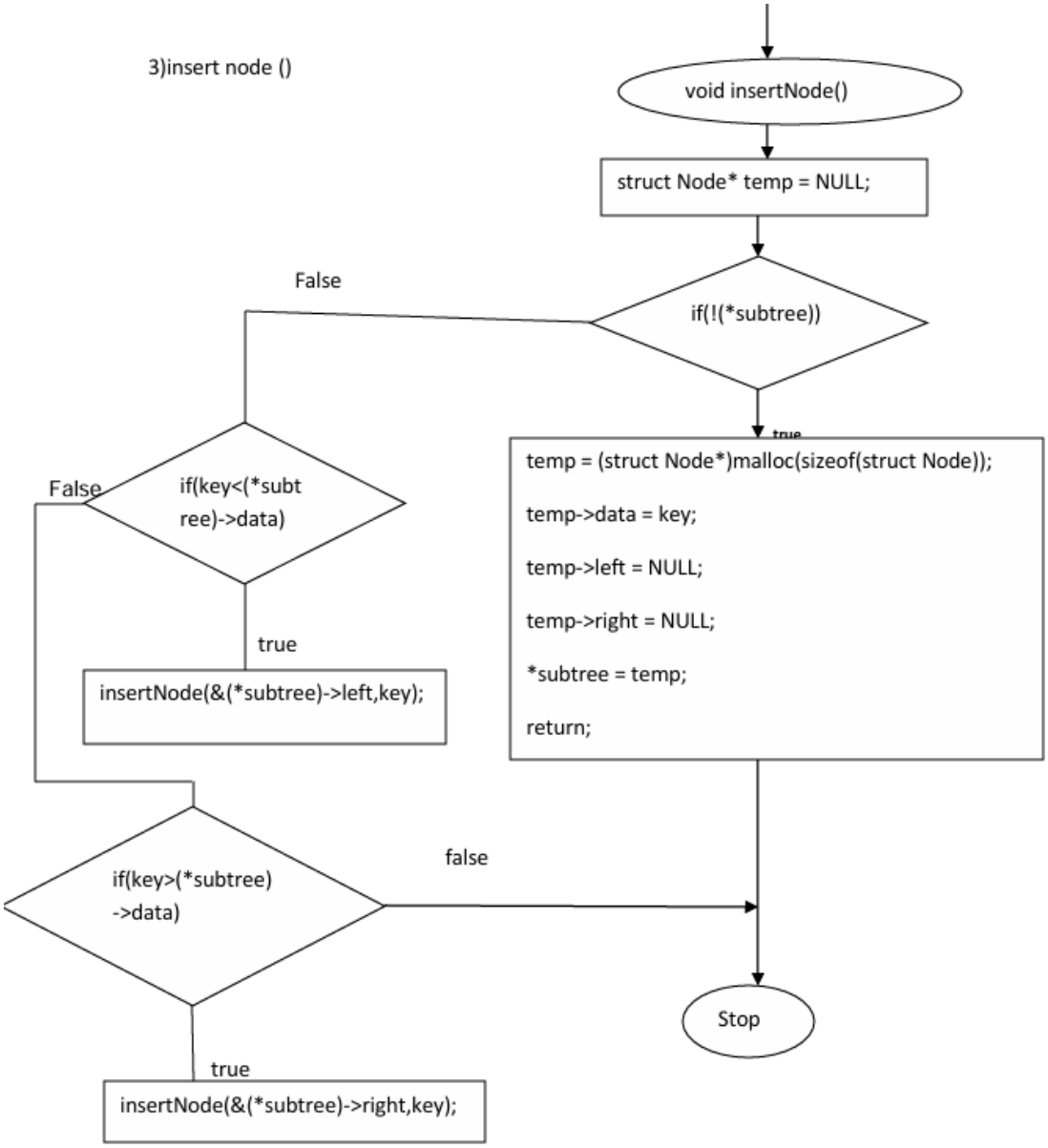


2)main function

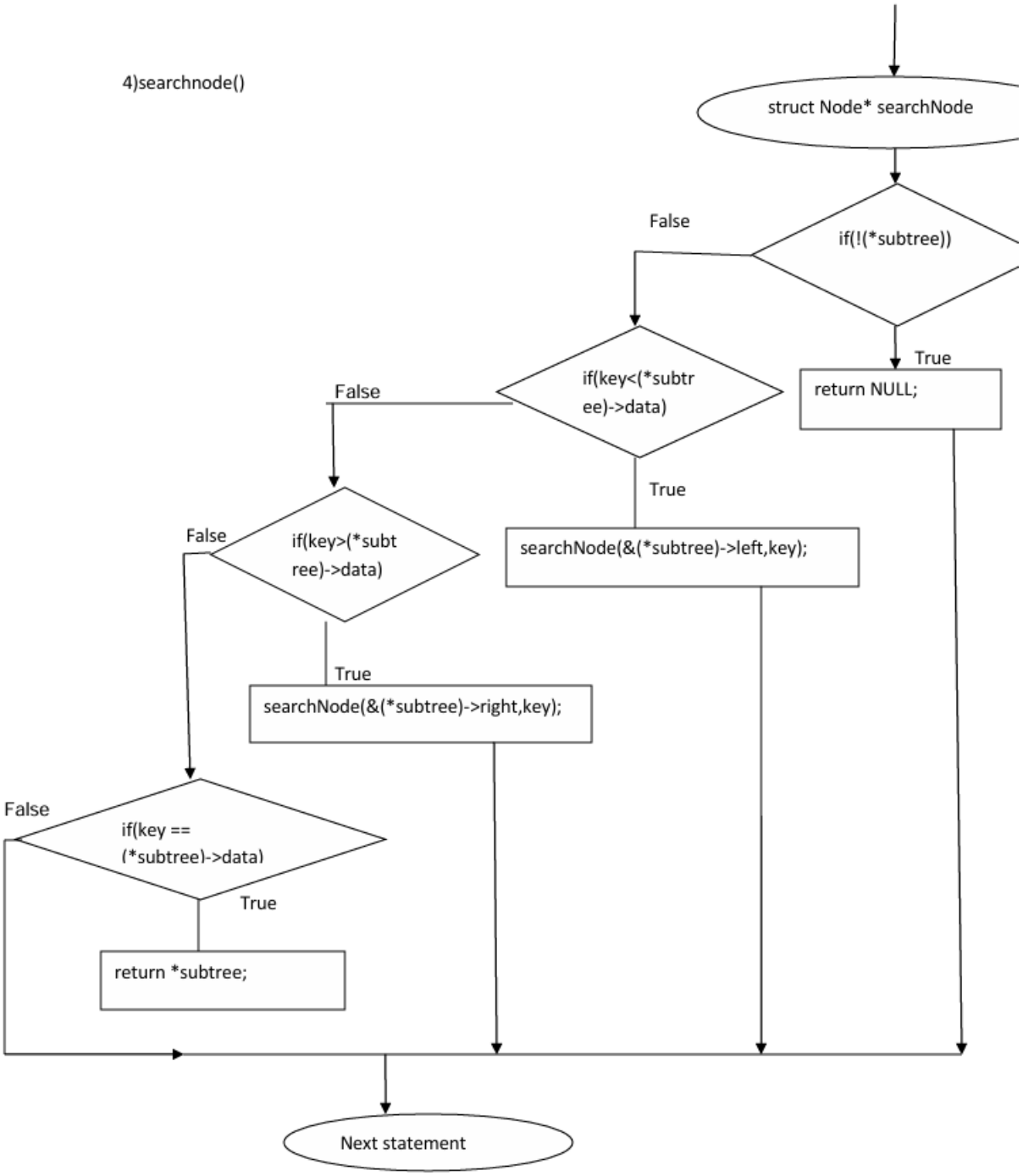




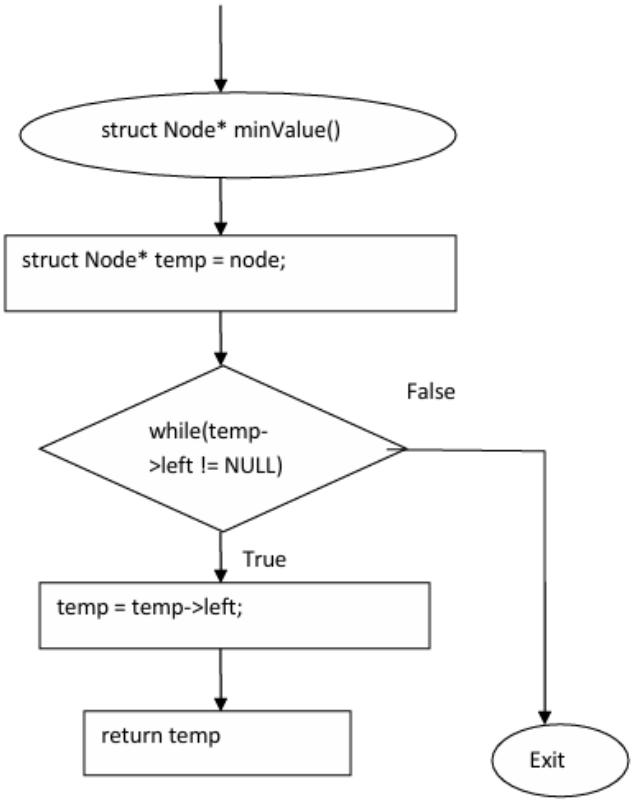
3)insert node ()



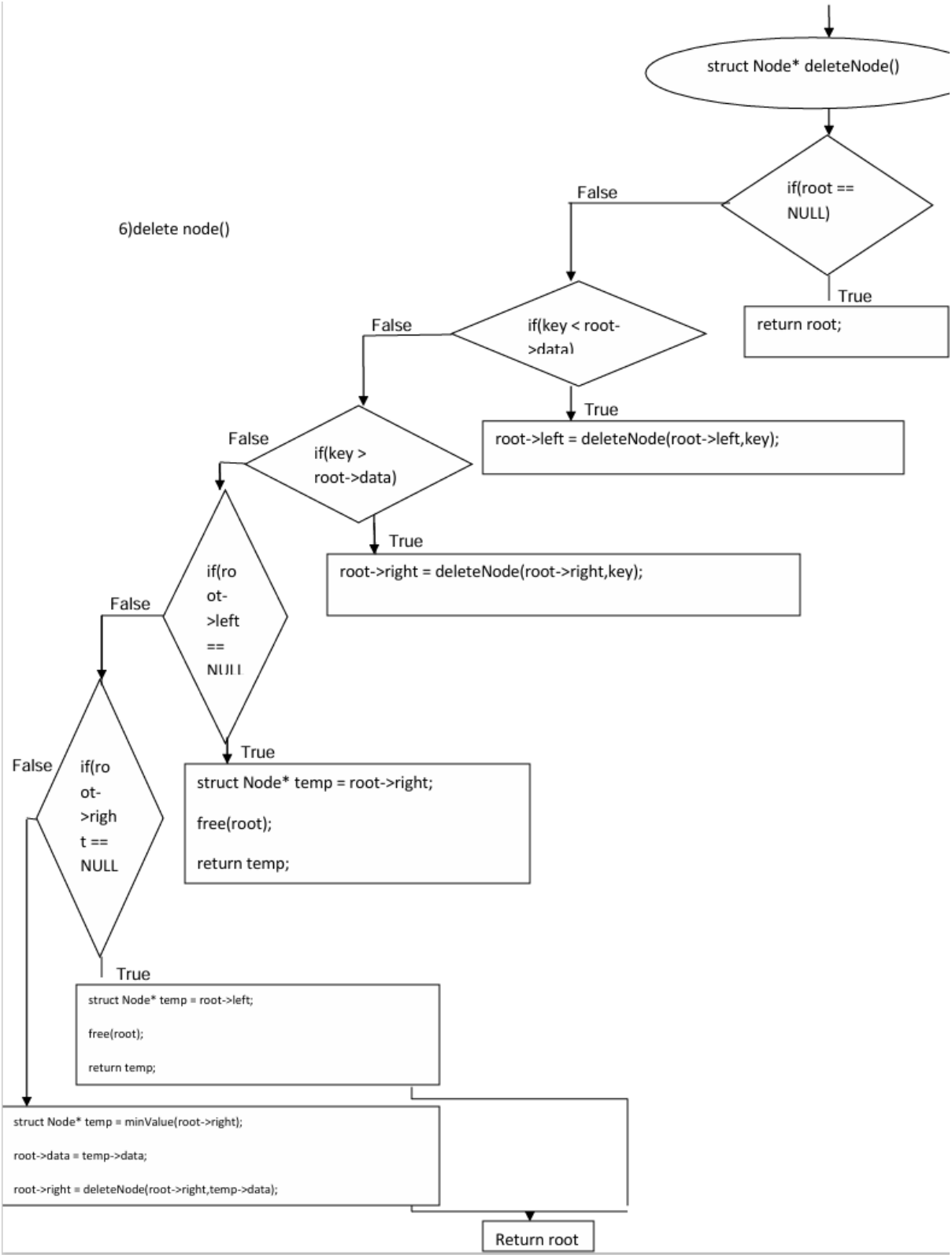
4)searchnode()



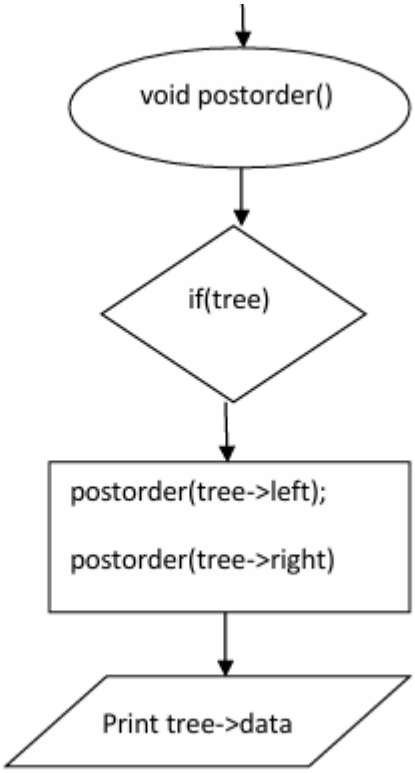
5)min value()


struct Node* minValue()
struct Node* temp = node;
while(temp->left != NULL)
False
True
temp = temp->left;
return temp
Exit

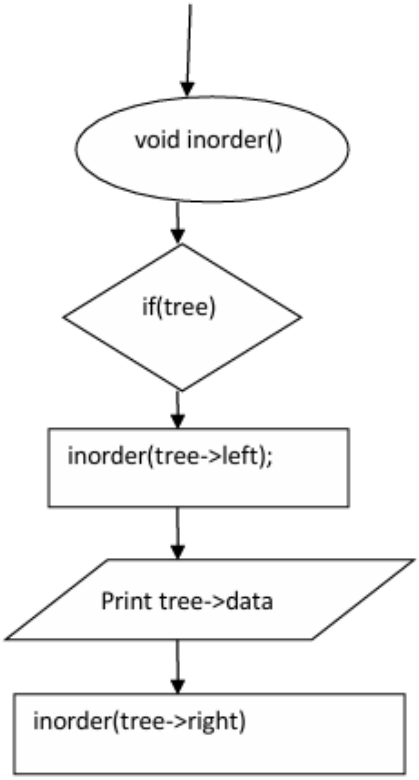
6)delete node()



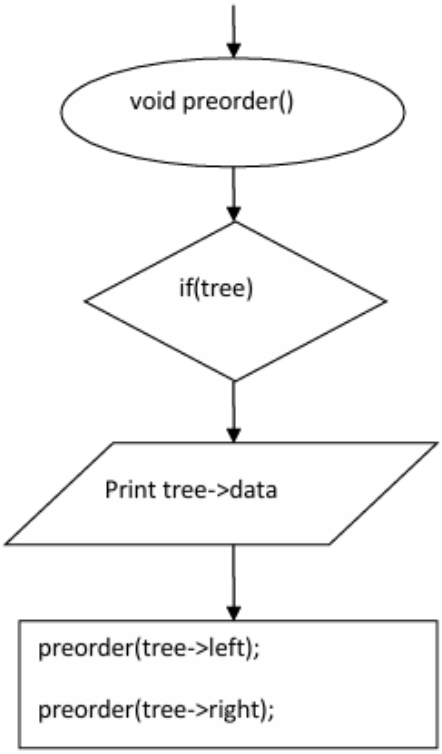
9)postorder()


void postorder()
if(tree)
postorder(tree->left);
postorder(tree->right)
Print tree->data

7)inorder()



8)preorder()

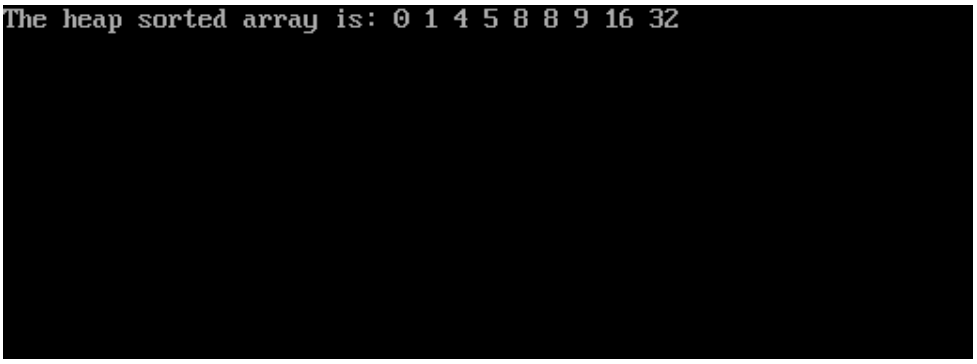


15. Write a program to Sort the following elements using heap sort (9.16,32,8,4,1,5,8,0).

```
#include<stdio.h>
void heapify(int array[],int size, int root)
{
// Find largest amoung root, left child and right child
int temp;
int largest = root;
int left = 2*root+1;
int right = 2*root+2;
if(left<size && array[left] > array[largest])
largest = left;
if(right<size && array[right] > array[largest])
largest = right;
//swap and continue heapifying if root is not largest
if(largest != root)
{
temp = array[root];
array[root] = array[largest];
array[largest] = temp;
heapify(array,size,largest);
}
}
void heapsort(int array[],int size)
{
//Build max heap
int i,temp;
for(i=size/2-1;i>=0;i--)
heapify(array,size,i);
for(i=size-1;i>0;i--){
temp = array[0];
array[0] = array[i];
```

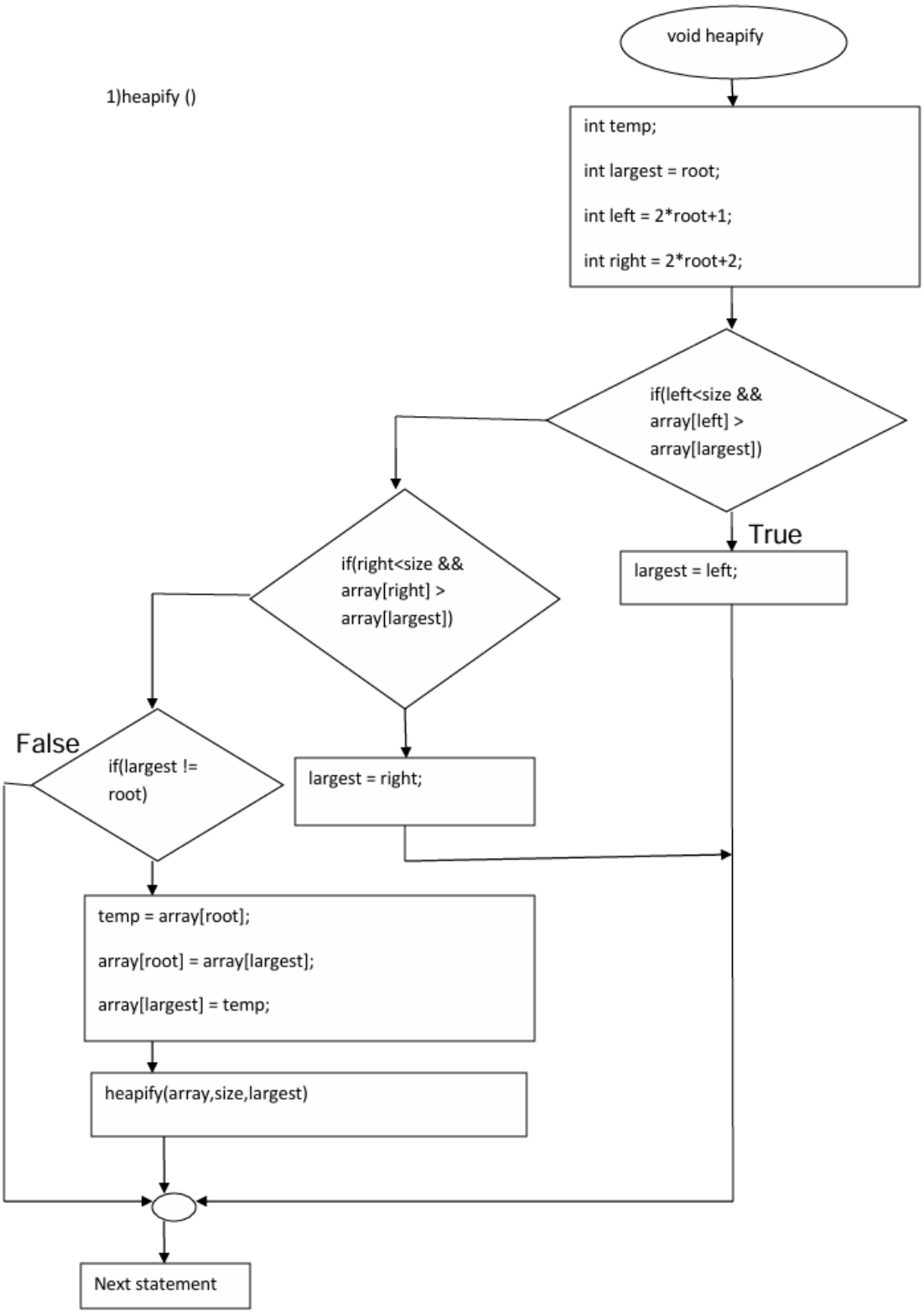
```
array[i] = temp;
//Heapify root element to get highest element at root again
heapify(array,i,0);
}
}

int main()
{
int j,arr[] = {9,16,32,8,4,1,5,8,0};
clrscr();
heapsort(arr,9);
printf("The heap sorted array is: ");
for(j=0;j<9;j++)
printf("%d ",arr[j]);
return 0;
}
```

Output:

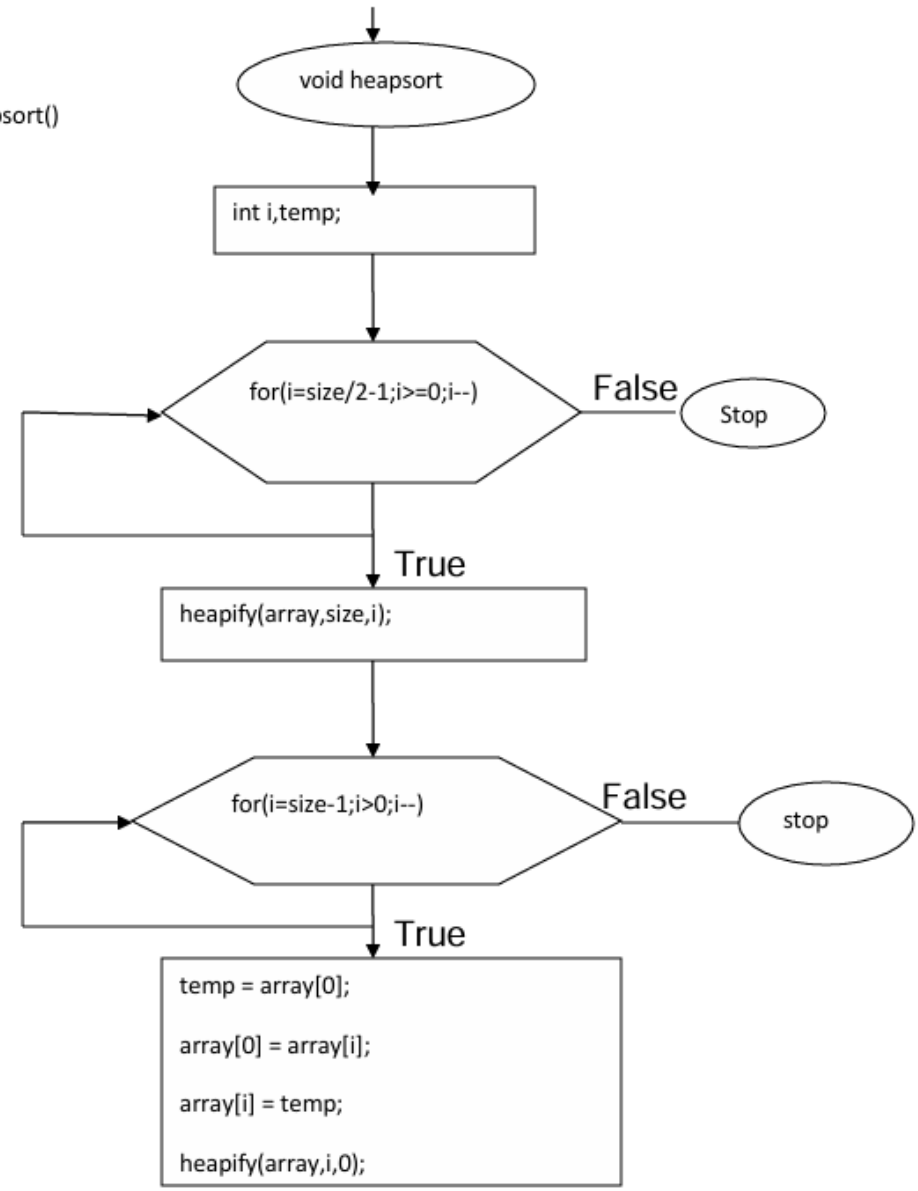


Flowchart:

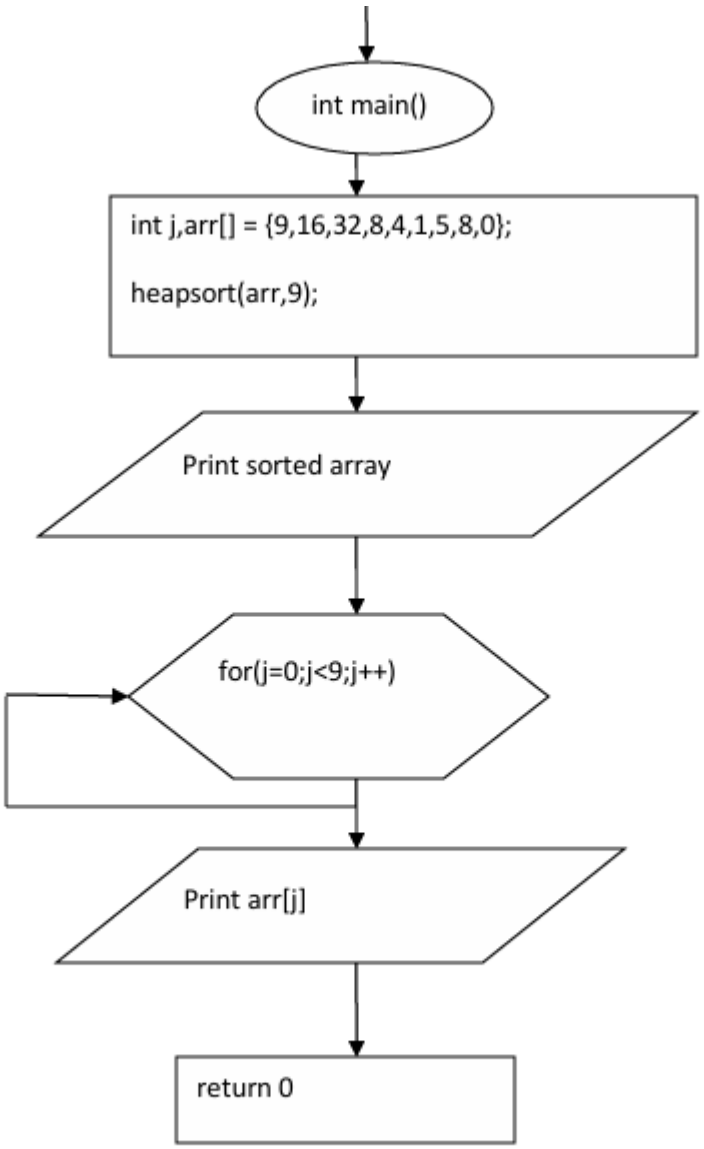
1)heapify ()


void heapify
int temp;
int largest = root;
int left = 2*root+1;
int right = 2*root+2;
if(left<size &&
array[left] >
array[largest])
True
largest = left;
if(right<size &&
array[right] >
array[largest])
False
if(largest !=
root)
largest = right;
temp = array[root];
array[root] = array[largest];
array[largest] = temp;
heapify(array,size,largest)
Next statement

2)heapsort()


void heapsort
int i,temp;
for(i=size/2-1;i>=0;i--)
False
Stop
True
heapify(array,size,i);
for(i=size-1;i>0;i--)
False
stop
True
temp = array[0];
array[0] = array[i];
array[i] = temp;
heapify(array,i,0);

3)main()


int main()
int j,arr[] = {9,16,32,8,4,1,5,8,0};
heapsort(arr,9);
Print sorted array
for(j=0;j<9;j++)
Print arr[j]
return 0

17] String operations: length, concatenation, substring extraction, and replace.

```
/* Given S1={“Flowers”} ; S2={“are beautiful”} I. Find the length of S1 II. Concatenate S1 and S2 II
I.
Extract the substring “low” from S1 IV. Find “are” in S2 and replace it with “is”  */
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void main()
{
char s1[10]="Flower",s2[15]="are beautiful";
int  length ,c=0,pos=1,i;
char s3[20],s4[5]="are",s5[5]="is",s6[10]=" beatiful";
clrscr();

length=strlen(s1);  //to find length
printf("\n Length of string s1 is %d",length);
printf("\n Concatination of s1 and s2 is %s",strcat(s1,s2)); //to concat strings

while(c<3)         //to find substring
{
s3[c] = s1[pos+c];
c++;
}
s3[c]='\0';

printf("\n Extracted string of s1 is %s",s3);

for(i=0;i<=3;i++)   //replace process
{
       if(s2[i]==s4[i])
       {
       s2[i]=s5[i];
       }
 }

printf("\n Replaced string  %s",strcat(s2,s6));
getch();
}
```

Program 18. Wrire a program to insert /retrieve an entryinto hash/from a hash table with open addressing using Linear probing.

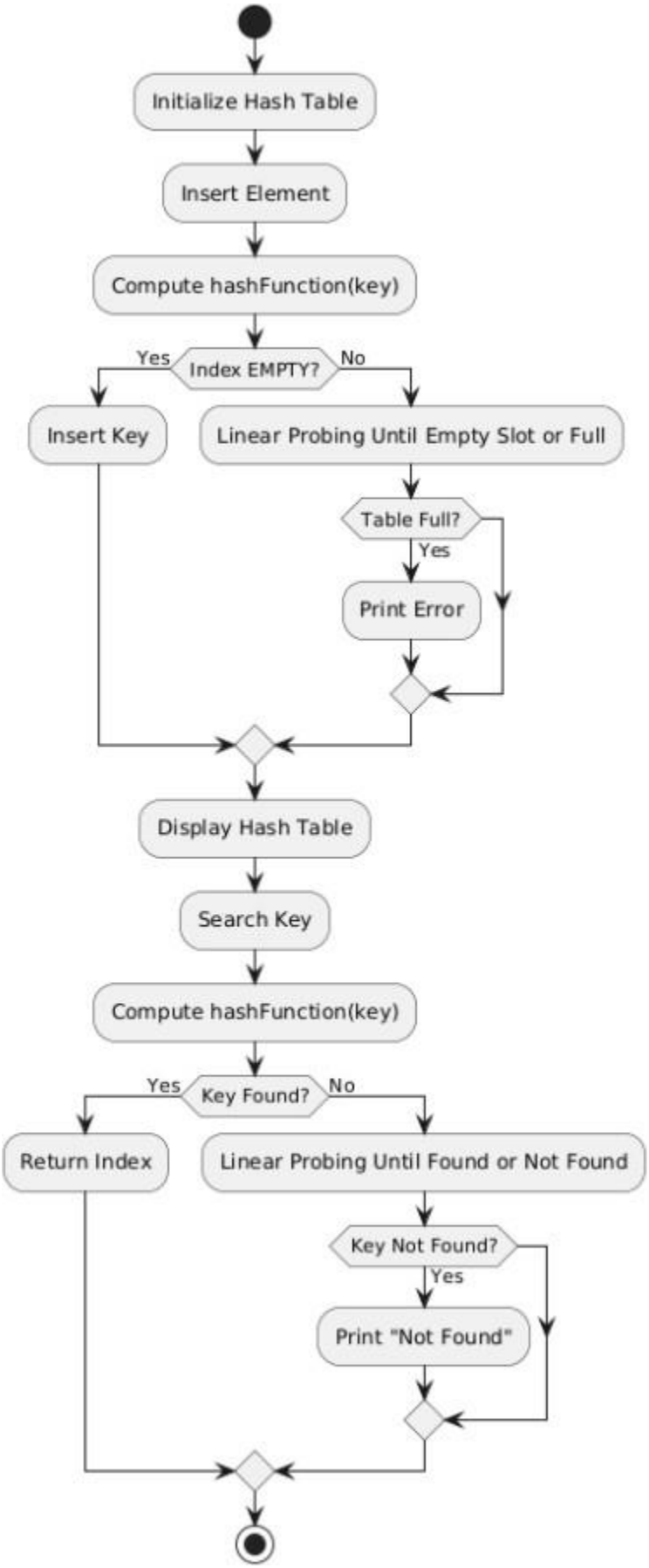
```
#include <stdio.h>
#define SIZE 10  // Define the size of the hash table
#define EMPTY -1

int hashTable[SIZE];  // Hash table initialization

// Hash function
int hashFunction(int key) {
   return key % SIZE;
}

// Insert function using linear probing
void insert(int key) {
   int index = hashFunction(key);
   int originalIndex = index;

   while (hashTable[index] != EMPTY) {
      index = (index + 1) % SIZE;
      if (index == originalIndex) {  // Table is full
         printf("Hash table is full, cannot insert %d\n", key);
         return;
      }
   }
   hashTable[index] = key;
   printf("Inserted %d at index %d\n", key, index);
}

// Search function using linear probing
int search(int key) {
   int index = hashFunction(key);
   int originalIndex = index;
```

```
    while (hashTable[index] != EMPTY) {
        if (hashTable[index] == key) {
            return index;
        }
        index = (index + 1) % SIZE;
        if (index == originalIndex) {  // Element not found
            break;
        }
    }
    return -1;  // Not found
}

// Display hash table
void display() {
    printf("\nHash Table:\n");
    for (int i = 0; i < SIZE; i++) {
        if (hashTable[i] != EMPTY)
            printf("Index %d: %d\n", i, hashTable[i]);
        else
            printf("Index %d: EMPTY\n", i);
    }
}

int main() {
    for (int i = 0; i < SIZE; i++) {
        hashTable[i] = EMPTY;  // Initialize hash table
    }

    insert(10);
    insert(20);
    insert(30);
    insert(25);
```

```
    insert(35);

    display();

    int key = 25;
    int position = search(key);
    if (position != -1)
        printf("\nElement %d found at index %d\n", key, position);
    else
        printf("\nElement %d not found in hash table\n", key);

    return 0;
}
```

output

```
Inserted 10 at index 0
Inserted 20 at index 0
Inserted 30 at index 1
Inserted 25 at index 5
Inserted 35 at index 6

Hash Table:
Index 0: 20
Index 1: 30
Index 2: EMPTY
Index 3: EMPTY
Index 4: EMPTY
Index 5: 25
Index 6: 35
Index 7: EMPTY
Index 8: EMPTY
Index 9: EMPTY

Element 25 found at index 5
```


Initialize Hash Table
Insert Element
Compute hashFunction(key)
Index EMPTY?
Yes
No
Insert Key
Linear Probing Until Empty Slot or Full
Table Full?
Yes
Print Error
Display Hash Table
Search Key
Compute hashFunction(key)
Key Found?
Yes
No
Return Index
Linear Probing Until Found or Not Found
Key Not Found?
Yes
Print "Not Found"